Don’t Overthink It Grids

BY CHRIS COYIER ON AUGUST 14, 2012

CSS, SASS

The vast majority of websites out there use a grid. They may not explicitly have a grid system in place, but if they have a "main content area" floated to the left a "sidebar" floated to the right, it's a simple grid.

If a more complex layout presents itself, people often reach for a grid framework. They assume grids are these super difficult things best left to super CSS nerds. That idea is perpetuated by the fact that a lot of the grid systems they reach for *are* very complicated.

> **HEY!**
>
> Note that this article was published in 2012. Floats was still the primary method for a grid system, and this article focuses on *really* simple methods for that, like just floating four 25% elements and not getting crazy with math and exotic gutter systems. These days, I'd highly recommend using CSS grid or flexbox for your grid system, if you even need to create an abstracted grid at all. It's arguably even easier and definitely more flexible and powerful.

Here's how I build grids. It's not hard or complicated. Even making them flexible is no big deal.

Context

A block level element is as wide as the parent it's inside (`width: auto;`). We can think of it as 100% wide. The wrapper for a grid probably don't have much to do with semantics, it's just a generic wrapper, so a `div` is fine.

HTML

```
<div class="grid">
  <!-- 100% wide -->
</div>
```

grid context

Columns

Let's start with a practical and common need: a main content area being 2/3 the width and a sidebar being 1/3 the width. We just make two column divs with appropriate class names.

HTML

```
<div class="grid">
  <div class="col-2-3">
     Main Content
  </div>
  <div class="col-1-3">
     Sidebar
  </div>
</div>
```

To make them next to each other, we just need to float them and apply widths. We can select both like this:

CSS

```
[class*='col-'] {
  float: left;
}
```

and individual width like this:

CSS

```
.col-2-3 {
  width: 66.66%;
}
.col-1-3 {
  width: 33.33%;
}
```

That's the whole premise of not overthinking grids.

Clearing Context

The parent element will collapse to zero height since it has only floated children. Let's fix that by clearing it. These days all you need is this:

CSS

```
.grid:after {
  content: "";
  display: table;
  clear: both;
}
```

Gutters

The hardest part about grids is gutters. So far we've made our grid flexible by using percentages for widths. We could make the math all complicated and use percentages for gutters as well, but personally I don't like percentage gutters anyway, I like fixed pixel size gutters. Plus, we're trying to keep too much thinking out of this.

The first step toward this is using `box-sizing: border-box;` . I like using it on absolutely everything.

CSS

```
*, *:after, *:before {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
```

```
  box-sizing: border-box;
}
```

Now when we set a width, that element *stays* that width, despite padding or borders being applied.

The second step is applying a fixed padding to the right side of all columns except the last one.

CSS

```
[class*='col-'] {
  padding-right: 20px;
}
[class*='col-']:last-of-type {
  padding-right: 0;
}
```

That's all there is to basic gutters.

Outside Gutters

Need gutters on the outside? I like using an opt-in class for this:

HTML

```
<div class="grid grid-pad">
  Grid with outside gutters also
</div>
```

Step one is to add left padding to the grid parent (and optionally top and bottom padding):

CSS

```
.grid-pad {
  padding: 20px 0 20px 20px;
}
```

Step two is to restore the right padding to the last column:

CSS

```
.grid-pad > [class*='col-']:last-of-type {
  padding-right: 20px;
}
```

More Column Choices

Super easy:

CSS

```
.col-1-2 {
  width: 50%;
}
.col-1-4 {
  width: 25%;
}
.col-1-8 {
  width: 12.5%;
}
```

Do whatever you want. Just make sure the column fractions add up to 1. Yeah, a little thinking, but easier than usual.

Sass

I'm not using it heavily here, but the whole bit becomes even a bit more succinct with SCSS/Compass:

scss

```scss
* {
  @include box-sizing(border-box);
}

$pad: 20px;

.grid {
  background: white;
  margin: 0 0 $pad 0;

  &:after {
    /* Or @extend clearfix */
    content: "";
    display: table;
    clear: both;
  }
}

[class*='col-'] {
  float: left;
  padding-right: $pad;
  .grid &:last-of-type {
    padding-right: 0;
  }
}
.col-2-3 {
  width: 66.66%;
}
.col-1-3 {
  width: 33.33%;
}
.col-1-2 {
  width: 50%;
}
.col-1-4 {
```

```
  width: 25%;
}
.col-1-8 {
  width: 12.5%;
}

/* Opt-in outside padding */
.grid-pad {
  padding: $pad 0 $pad $pad;
  [class*='col-']:last-of-type {
    padding-right: $pad;
  }
}
```

Modules

I like to work within these grids with "modules".

HTML

```
<div class="grid">
  <div class="col-2-3">
     <article class="module">
        stuff
     </article>
     <article class="module">
        stuff
     </article>
  </div>
  <div class="col-1-3">
    <aside class="module">
       Sidebar stuff. Sub modules?
    </aside>
  </div>
</div>
```

It feels nice breaking up content into bits this way. The bonus side effect being that each module can have padding of it's own, keeping text away from the edges of the grid.

Result

Here's a demo on CodePen.

Browser Whatnot

Works just great in IE 8 and up and all the other standard stuff. If you need IE 7 support, you'll have to do something else =).

Also, Flexbox is going to make this even easier and better (in various ways, including reordering on demand), but I think we need about a year until we can start to think about using it.

Related

Check out OOCSS grids.

Related

Comments

eQRoeil

AUGUST 14, 2012

There is a polyfill for box-sizing https://github.com/Schepp/box-sizing-polyfill
Source @goetter http://www.knacss.com/ (same grid with float and other option with table-cell)

Paweł P.

AUGUST 15, 2012

Hmmm... **Interesting!**

Ramveer

AUGUST 15, 2012

Nice

Patrick Laughlin

AUGUST 21, 2012

Or you could use a mixin if you're into that kinda thing. I'm sure this could be written better. =)

```
@mixin box-model($max_height: auto, $max_width: auto,
    $plength: length($padding);
    @if $plength == 4 {
        $ptop: nth($padding, 1);
        $pright: if($plength >= 2, nth($padding, 2), $
        $pbottom: if($plength >= 3, nth($padding, 3),
        $pleft: if($plength >= 4, nth($padding, 4), $p
    }
    @if type-of($max_height) == number { height: $max_
    padding: $ptop $pright $pbottom $pleft;
```

```
    @if type-of($max_width) == number { width: $max_w
}
```

arf_root

AUGUST 14, 2012

What I wonder about this example, and a lot of the grid frameworks, is the side effects of the naming scheme. In this example you have a bunch of class names that effectively have the presentation encoded into their names (col-1-3, col-2-3, etc). What happens when you have to create a new stylesheet for a user agent with a much narrower screen? Now you have class names that make no semantic sense… 'col-1-3' might be 100% width.

Class naming is always a constant battle for me. When I try going ivory tower, pure semantic names I end up with a bunch of classes/ids that are so generic the stylesheet is impossible to interpret.

Atimoda

AUGUST 14, 2012

You dont need to use these names, you can use your own semantic class name, like "menu" and "entries", and then you set in css how large are the columns of everything, like:

```
.col-1-3,
.menu,
.sidebar { width: 33%; }

.col-2-3,
.entries,
.comments { width: 66%; }

.col-3-3,
.header,
.footer { width: 100%; }
```

this is the way i use it to keep html semantic.

Chris Coyier

AUGUST 14, 2012

Which is a classic use case of @extend =)

Balazs Sziklai

AUGUST 15, 2012

@Chris Exactly!

I have no problems shoving non-semantic class names into my markup but if you itchy about that just extend your semantic class/id to the grid class in sass.

```
header{
  padding: 1em 0;
  @extend .col-3-3;
}
```

AntoxaGray

AUGUST 15, 2012

I never been confused that all grid classes on mobile have 100% width. Well… because they are all 100%?

Joe Snell

AUGUST 17, 2012

Uhm, hello! First I've seen @extend with SASS…. where have I been?! Going to deploy now! Thanks!

Matt Copeland

AUGUST 20, 2012

@joe be careful with @extend if you're using media queries.

Gotchas:
http://designshack.net/articles/css/sass-and-media-queries-what-you-can-and-cant-do/

SASS 3.2 media queries and @extend:
http://thesassway.com/intermediate/responsive-web-design-in-sass-using-media-queries-in-sass-32

Eelco

SEPTEMBER 23, 2012

Problem with @extend here is that it won't use "[class*='col-']". So you'll be missing your "float: left;". Just a quick heads up.

Kevin

SEPTEMBER 29, 2012

Wouldn't the new %placeholder syntax have great use here to not include the column classmates in your code but continue utilize them to ensure you're adhering to your grid (assuming you don't need to dynamically create columns for the purposes of the site you're using the grid for, though you can still use the classnames and just create the equivalent %placeholder) to lower the number of selectors you use on your stylesheet significantly?

Ryan

AUGUST 14, 2012

Grid systems that depend on specific (and often overly verbose) HTML drive me nuts. I am pretty hardcore in the belief that all class names should be reflective of the type of data they contain and that any layout driven code should strictly remain within the CSS file. However, HTML grid systems completely undermine this and in a way remind me of the days back when we used tables for layout.

Jim Houx

SEPTEMBER 23, 2012

I'm kind of newbie with this stuff, but the problem I see is that presentation and content really aren't 100% separable. What if I display same types of content differently on different pages? Now I've got to define more selectors in the stylesheet for per-page cases. Whereas, if I'm just doing a unique page and I want to arrange content in a certain way, I can use specific grid classes to do so right in the markup without having to go pick apart a stylesheet. If you try to 100% abstract presentation from content, what you're targeting is only a subset of use cases. There's no single solution for every design problem.

Donald Allen

AUGUST 14, 2012

Is using *:after, *:before required? I've always used just *.

Chris Coyier

AUGUST 14, 2012

See: https://twitter.com/garazi/statuses/234076720595075072

jochem

AUGUST 14, 2012

Most grid systems arent any morercomplicated then this. Whats the difference between div-1-3 / div-2-3 and span4 / span8 ? Basicly its the same. Except perhaps the pixel based gutter.its just that most grid systems are merely a smaller part of a larger css framework. But ive often just selected the items i needed and discarded everything else. The end effect is the same though.
I'm curious though. I see a lot of people adding a similar clearfix to their css. Why ?Almost everytime adding overflow hidden to the parent gets the same result without adding extra content we dont need.

Jim Houx

SEPTEMBER 23, 2012

I wish someone had replied to jochem's question. I was wondering the exact same thing. Any help?

TheMilkman

NOVEMBER 28, 2013

adding overflow: hidden is a very easy and good way. but if you have some contents that need to be displayed in an overlapping way and are markupped in the same box (e.g. small layers/tooltips or validationbubbles), then you can't use overflow: hidden.

cheers

Nicolas Gallagher

AUGUST 14, 2012

Nice post, Chris.

I've got a partially finished (Sass-based) grid system that does some of these things, but uses `inline-block` to get some pretty handy layout benefits, and it makes an attempt to play well in the responsive design world – https://github.com/necolas/griddle

A few comments:

```
[class*='col-'] {
  float: left;
}
```

This is a substring match, which means it's not very robust. If you have any other class that contains `col-` , then the styles end up applied there too. Generally, I prefer to avoid this type of selector for components, especially in larger codebases.

```
.grid:after {
  content: "";
  display: table;
  clear: both;
}
```

Yeah, you can contain floats with just this code (you may as well use `display: block` in that case), but it results in different containment of top and bottom margins and inconsistency with alternative float containment methods like `overflow: hidden` . To avoid both those issues, you still need the `:before` pseudo-element too.

This isn't so much of an issue in a grid component, but for general float containment, I'd still recommend using `:before` .

Thanks.

Chris Coyier

AUGUST 14, 2012

Thanks for the updates.

An alternative to `[class*='col-']` could easily be just adding an additional class in the HTML, e.g. `class="col col-1-3"` . In my case I'm usually pretty intimate with my whole code base so I'm cool with the single class name, but I'd agree in larger less familiar group-ish code bases the second class is better.

So do you recommend using *both* `:before` and `:after` ?

Jim Houx

OCTOBER 2, 2012

Nicolas said: "for general float containment, I'd still recommend using :before."

Re: You're suggesting a float containment model that isn't compatible with IE 7 and 8, unless you intend on including the javascript hack that adds 'before' support for those browsers, but I suspect you'd get unpredictable behavior.

I highly recommend sticking to backwards compatible methods for things like float containment.